Linux Shell Scripts - Fundamentals

Shay Berman
shaybery@gmail.com

Publication date: 13/1/2018

Final Exercise

Contents

sync.sh

(30 points)
Write a bash script that sync source dir to destination dir.
Script requirements below:

1. Sync a fileX that does not exist in the destination or that has different content from the same fileX in destination. Don't copy a fileX if the same fileX already exists with the same content in destination.
2. Don't delete anything from the destination, just sync from source to destination.
3. The source may include regular files and sub directories. Make sure the sync is done on all sub directories.
4. Print each file that needs to be synced. In addition, print at the end of the script a summary that includes: number of files that were copied and the total number of files that were scanned.
5. Script should ignore SIGINT and SIGTERM.
6. The script should NOT use rsync or cp –u commands.
7. Set the Shebang of the script to be "#/bin/bash -e". So make sure you should handle every error right!
8. Script flags:
   i) flag "-s" <dir> : source dir.
   (if not given, then it must be given by environment variable SDIR)
   ii) flag "-d" <dir> : destination dir
   (if not given, then it must be given by environment variable DDIR)
   iii) flag "-a" :make sure the script also sync the file permission and ownership. (optional flag)
   iv) flag "-v" : The script should print every file that it go though, including files that doesn't need to be copy at all. (optional flag)
   v) flag "-y" ask "r u sure? (y\n)" before sync any file. (optional flag)
   vi) flag "-t" is a test mode, run the script but without doing any copy. (optional flag)
9. The script should also handle a "named pipe" file type. The script should automatically create the pipe file in the destination file.

SystemStatus.sh

(30 points)

Improve the systemStatus.sh script by adding more system verification and improvements as follow:

1. Check file-systems size usage, alert if there is a file system with more than 90% usage.
2. Alert if inodes usage is higher than 90% on a file system.
3. Alert if there are zombie processes.
4. Alert if there are more than 20 ports in listen mode.
5. Alert if number of installed rpms is higher than 3200
6. Alert if there are more than 100 docker containers running.
7. Refactor the script:
    a. Make each verification as a function.
    b. The script should print which test named failed and which tests passed.
    c. Instead of hardcoded thresholds, the script should get a config file with all the thresholds (get it with –c <file> flag). Every line in the file should look like this: [verification function name] [> or <] [value]. Here is an example:

```
cpu_usage > 90%
swap_usage > 80%
zombiees_processes > 0
```

    The script should ignore comments or empty lines from the thresholds config file, also ignore unknown thresholds with warning.

docker_clean.sh

(20 points)

Develop a bash script that cleans up docker containers according to a given state, or docker images according to regex pattern. Here is the usage of the script:
docker_clean [-c ps -s <status> | -c images -o <pattern>] [-t]

-c ps -s STATUS : remove all the docker containers with a given status. Status can be “running” or “exited”.
-c images -o pattern : remove all the images that match the pattern.
-t : a test mode, don’t delete anything just show what will be deleted

DevOps

(20 points)

1. Draw an extensive CI pipeline for your application. Provide CI pipeline with as many jobs needed to make sure your application is ready for production. Just provide the job name (with 1 sentence description of the job) and link to the job that it triggers. Pay attention that your application is written in Python, placed in github and is delivered as Docker image.
2. What problems Agile tries to solve? (short sentence)
3. What problems DevOps tries to solve? (short sentence)